Charlie Basantes

COP 4045 001

2/12/20

Homework #2

1. a) Write a function called line_number that takes as parameters two strings representing file names. Assume these are text files. The function reads the file indicated by the first parameter and writes its lines prefixed by the line number to the file represented by the second parameter. The function must have a proper docstring and annotations. Use try/except and in case of an error print a user-friendly message to the terminal and re-raise the exception.

   b) Write a function called parse_functions that takes as parameter a string representing the name of a .py file. The function reads and parses the Python file and returns a tuple of tuples where each tuple has its element 0 a function name, element 1 the line number, and element 2 the function code as a string (signature and body), with all comments removed. The top-level tuple returned must be ordered alphabetically by the function name. Function parse_functions must have a proper docstring and annotations. Use try/except and in case of an error print a user-friendly message to the terminal and re-raise the exception. Write in the main function code that calls parse_functions on the problem 1 Python file and displays the returned tuple.

My solution:

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb  5 20:21:27 2020

@author: solid
"""

'''
This function reads the file indicated by the first parameter and writes its
lines prefixed by the line number to the file represented by the second
parameter
'''

def line_number(file1,file2):

  f1=open(file1, "r")
  f2 = open(file2, "w")
  try:
      for x in f1.readlines():
          f2.write(x)
      f2.close()
  except:
```

```
      print("text file is empty,or does not exist.")

def parse_functions(py_file):
    function_list=[]
    try:
        with open(py_file,"r") as read_file:
            line_number = 1
            function_name =''
            function_body =""
            for line in read_file.readlines():
                if "def" in line:

                    if("#" in line):
                          tup=line_number,function_name,function_body
                          print(tup)
                          function_list.append(tup)
                    else:
                          function_body=""

                    function_name =
line[line.index("def")+3:line.index('\n')].strip()
                    function_body = line

                else:
                    function_body+=line
                line_number+=1
            tup=line_number,function_name,function_body
            function_list.append(tup)
        return tuple(function_list)
    except:
        print('Unable to read/write the file:')



def main():
    #got this to work
    line_number('test.txt', 'test.py.txt')
    #thinks every function is main...
    print(parse_functions('test.py'))


main()
```

2. a) Write a list comprehension that returns all tuples (a,b,c), with a,b,c integers, such that 1<=a,b,c<=100 and a2+b2=c2 .

   b) Consider a list of strings, like this: ['one', 'seven', 'three', 'two', 'ten']. Write a list comprehension that produces a list with tuples where the first element of the tuple is the length of an element in the initial list, the second element of the tuple is the element of the initial list capitalized, and the resulting list contains only tuples for strings with the length longer than three characters. For our example the list comprehension should return [(5, 'SEVEN'), (5, 'THREE')].

   c) Consider a list of full names (formatted “Firstname Lastname”), like["Jules Verne", "Alexandre Dumas", "Maurice Druon"]. Write a list comprehension that produces a list with the full names in this format: “Lastname, Firstname”. The resulting list should look like ['Verne, Jules', 'Dumas, Alexandre', 'Druon, Maurice']. The simplest solution may involve a nested comprehension: [….
   for … in [… for … in …]].

   d) Write a function called concatenate that takes as parameter a separator (a string) and an arbitrary number of additional arguments, all strings, and that returns the concatenation of all given strings using the given separator. Example: concatenate(‘: ‘, “one”, “two”, “three”) returns “one: two: three” and concatenate(‘ and ‘, “Bonny”, “Clyde”) returns “Bonny and Clyde”. For a single string we have: concatenate(‘ and ‘, “single”) return “single”.

   My solution(adjusted):

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb  5 22:29:31 2020

@author: solid
"""
#import math

'''
```

```
a) Write a list comprehension that returns all tuples (a,b,c), with a,b,c
integers,
such that 1<=a,b,c<=100 and a2+b2=c2
 '''
#nested for loops to go through each value from 1-100 and checks if a^2+b^2 =
c^2. otherwise run until loop ends
list_comprehension_a = [ (a, b, c) for a in range(1,101)  for b in
range(1,101)  for c in range(1,101)  if((a*a + b*b) == (c*c)) ] #simplified
print(list_comprehension_a)
#to space out each part
print("")


'''
b) Consider a list of strings, like this: ['one', 'seven', 'three', 'two',
'ten']. Write a list comprehension that
produces a list with tuples where the first element of the tuple is the
length of an element in the initial
list, the second element of the tuple is the element of the initial list
capitalized, and the resulting list
contains only tuples for strings with the length longer than three characters
'''

strings_list = ['one', 'seven', 'three', 'two', 'ten' ,]
#finds the length of the string array, capitalizes very character. then
checks the length of each word if it
#is greater than 3 characters and prints the length of the string array, and
the element with more than 3 characters
list_comprehension_b = [ (len(strings_list[i]), strings_list[i].upper()) for
i in range(len(strings_list)) if len(strings_list[i]) > 3]
print(list_comprehension_b)
#to space out each part
print("")


'''
c) Consider a list of full names (formatted “Firstname Lastname”),
like["Jules Verne", "Alexandre
Dumas", "Maurice Druon"]. Write a list comprehension that produces a list
with the full names in this
format: “Lastname, Firstname”.
'''

name_list = ["Jules Verne", "Alexandre Dumas", "Maurice Druon"]
#go through the string and look for the space between the first and last
name. place evreything after the space first, place a comma and then place
what was previously in front of the space next
#that was the last name is displayed then the first name is displayed: (last,
first). since using a single list using a for-in loop to go through the whole
list
list_comprehension_c = [adjust_name[adjust_name.index(' ') + 1:] + ',' +
adjust_name[0:adjust_name.index(' ')] for adjust_name in name_list]
```

```
print(list_comprehension_c)
#to space out each part
print("")

'''
d) Write a function called concatenate that takes as parameter a separator (a
string) and an arbitrary
number of additional arguments, all strings, and that returns the
concatenation of all given strings using
the given separator.
'''

def concatenate(seperator, *string_list):
    list_plus_seperator = seperator.join(string_list)
    #print(list_plus_seperator)
    return list_plus_seperator #returns now

#prints them here now instead of in function
print(concatenate(': ', "one", "two", "three"))
print(concatenate(' and ', "Bonny", "Clyde"))
print(concatenate(' and ', "single"))
```

3. For the following, all functions must have a proper docstring and annotations. Use try/except and in case of an error print a user-friendly message to the terminal and raise or re-raise the exception. For this problem we will develop a contact list, similar to what is on smartphones. The contact list data structure is a list of tuples with the following format (name, nickname, phone#). For instance, a sample contact list could be [("Beyonce Knowles", "bey", "561-1234321"), ("Cardi B", "Belcalis", "305-4399521"), ("Earl Simmons", "DMX", "305-1010101")]. The contact list must be stored in a Python list of tuples, as above, always sorted alphabetically on the name field. An empty contact list is represented by the empty list []. For your solution to get any credit do NOT use dictionaries.

a) Write a function that adds a contact to a contact list. If the contact name existed before in the list then it will be changed to the new entry and the function returns False. Otherwise, the function should return True.

b) Write a function that removes a contact from a contact list. If the contact name existed before in the list then the function returns True. Otherwise, the function should return False.

c) Write a function that finds a contact tuple from a contact list by passing the contact name or contact nickname. Use default parameter values to deal with this choice. The function returns the tuple if the contact is found and returns None otherwise.

d) Write a function that saves a contact list to a .CSV file. The function takes as parameter the file name. The CSV file format must have name, nickname, phone# on a line for each contact in the list.
e) Write a function that reads a contact list from a .CSV file with the format described for part d). The function takes as parameter the file name and returns the contact list object (… sorted alphabetically). f) Write a main function that tests all the functions above.

g) EXTRA CREDIT: 5 points Write a function test where you use the testif function from Module 2 to test the functions written for parts a)-e).

my solution(fixed):

```
# -*- coding: utf-8 -*-
"""
Created on Thu Feb  6 19:06:25 2020

@author: solid
"""

'''
a) Write a function that adds a contact to a contact list. If the contact
name existed before in the list then
it will be changed to the new entry and the function returns False.
Otherwise, the function should
return True.
'''
#change, contacts being used as parameter
def add_contacts(name, nick_name, phone_number, contacts):

    for i in range(len(contacts)):
        #checking if anything is existed before, change it, return false
        if(contacts[i][0] == name or  contacts[i][1] == nick_name or
contacts[i][2] == phone_number):
            contacts[i] = (name,nick_name,phone_number)
            return False
    contacts.append((name, nick_name, phone_number))
    #sort alphabetically
    contacts.sort(key = lambda x: x[0])
    return contacts

'''
b) Write a function that removes a contact from a contact list. If the
contact name existed before in the
list then the function returns True. Otherwise, the function should return
False.
'''
#change, contacts being used as parameter
def remove_contacts(name, nick_name, phone_number, contacts):
    remove = 0
```

```
    contact_found = False
    #go through the list
    for i in range(len(contacts)):
        #see if contact already exists within the list and if found remove it
        if(contacts[i][0] == name or  contacts[i][1] == nick_name or
contacts[i][2] == phone_number):
            remove = i
            contacts.pop(remove)
            contact_found = True
        else:
            contact_found = False

    return contact_found

'''
c) Write a function that finds a contact tuple from a contact list by passing
the contact name or contact
nickname. Use default parameter values to deal with this choice. The function
returns the tuple if the
contact is found and returns None otherwise
'''
#None  is equivalent to a null ptr
def find_contact_byname_or_nickname(name = None, nick_name = None,
*contacts):
      if name is not None:
          for i in contacts:
              if name in i:
                  return i
          return i

      if nick_name is not None:
          for i in contacts:
              if nick_name in i:
                  return i
          return None

'''
d) Write a function that saves a contact list to a .CSV file. The function
takes as parameter the file
name. The CSV file format must have name, nickname, phone# on a line for each
contact in the list.
'''
#change, contacts being used as parameter so it know what it is
def save_csv_file(file_name, contacts):
    #open takes agrument file to open, and write indicated with w and writes
into the file
    file = open(file_name,"w")
    file.write("name, nick_name, phone_number\n")
    for i in contacts:
        #unsure how to use join() so left it alont since this still works
        file.write("{}, {}, {}\n".format(i[0], i[1], i[2]))
```

```
'''
e) Write a function that reads a contact list from a .CSV file with the
format described for part d). The
function takes as parameter the file name and returns the contact list object
(… sorted alphabetically).
'''
def read_csv_file(file_name):
    #Open takes agrument file name, and read indicated with r and reads the
contents of the filefile
    file = open(file_name, "r")
    read = file.readline()
    list_of_contacts = []

    while(read):
        list_of_contacts.append(tuple(read.split(",  ")))
        read = file.readline()
    #sort the list
    list_of_contacts.sort(key = lambda x: x[0])
    return  list_of_contacts

'''
f) Write a main function that tests all the functions above.
'''

def main():
    contacts = [] #no longer a global variable, whoops, fixed that, will be
added as extra parameter
    try:
        add_contacts("Earl Simmons", "DMX", "305-1010101",contacts)
        add_contacts("Cardie B", "Belcalis", "305-4399521",contacts)
        add_contacts("Beyonce Knowles", "bey" , "561-1234321",contacts)
        print("\n Contacts added successfully.")
        print(contacts)
    except:
        print("There was an error while adding a contact.\n")

    to_remove = input("Would you like to test remove contact from the list?:
")
    if(to_remove == 'yes' or to_remove == 'Yes'):
        try:
            remove_contacts("Earl Simmons", "DMX", "305-1010101", contacts)
            print("\n Contact was removed successfully.")
            print(contacts)
        except:
            print("There was an error when attempting to remove the
contact.\n")

    add_contacts("Earl Simmons", "DMX", "305-1010101", contacts)

    try:
        find_contact_byname_or_nickname("Beyonce Knowles", "bey", contacts)
        print("contact found")
```

```
        print(find_contact_byname_or_nickname("Beyonce Knowles", "bey",
contacts))
        save_csv_file("contacts_file.csv", contacts)
        print(read_csv_file("contacts_file.csv"))
    except:
        print("Unable to find the contact")

'''
testing all functions
'''
main()
```

4.

For this problem we work with CSV files and dictionaries using data from IMDB lists with top rated and top grossing movies. These CSV files are linked on the Homework 2 Canvas page:

- imdb-top-rated.csv, listing the ranking of the top rated 250 movies. It has this format: Rank,Title,Year,IMDB Rating
- imdb-top-grossing.csv, listing the ranking of the highest grossing 250 movies. It has this format: Rank,Title,Year,USA Box Office
- imdb-top-casts.csv, listing the director and cast for the movies in the above files. It has this format: Title,Year,Director,Actor 1,Actor 2,Actor 3,Actor 4,Actor 5. The actors are listed in billing order. This file does not have a heading. These files are from Duke University and seem to date from 2014.

a) Write a function called display_top_collaborations that displays the ranking of tuples (director, first billed actor (i.e. Actor1), and number of movies) for movies in which the director and actor worked together also listed in the top rated movie list. The list should be in descending ordered of the total number of movies that director and that actor worked together. For equal values the order does not matter. Notice that the number of movies in the imdb-topcasts.csv file exceeds the number of movies in imdbtop-rated.csv. You may have to open the csv files like this: open(filename, 'r', encoding = 'utf-8').

b) Write a function called display_top_directors that displays the ranking of movie directors from the top grossing list ordered by the total box office money they produced.

c) Write a main() function that tests the code from parts a) and b). Take a screenshot of the program's output (parts a, b, c) where each printed ranking list is limited to the first 5 entries and insert it in the doc file right after the code. You get 8 points deducted if the screenshot is missing.

My solution:

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 12 22:00:08 2020
```

```
@author: solid
"""

import pandas as pd

'''
a) Write a function called display_top_collaborations that displays the
ranking of tuples
(director, first billed actor (i.e. Actor1), and number of movies) for movies
in which the
director and actor worked together also listed in the top rated movie list.
The list should be in descending ordered of the total number of movies that
director and
that actor worked together. For equal values the order does not matter.

'''
def display_top_collaborations():
    #read excel files
    top_casts = pd.read_csv('imdb-top-casts.csv',header=None)
    top_casts.columns = ['Title','Year','D','A1','A2','A3','A4','A5']
    top_rated = pd.read_csv('imdb-top-rated.csv')
    cast_list = dict()

    for index, row in top_rated.iterrows():
        movie_title = row['Title']
        find_smae = top_casts.loc[top_casts['Title'] == movie_title]
        if ((find_smae.iloc[0]['D'],find_smae.iloc[0]['A1']) in cast_list):
            cast_list[(find_smae.iloc[0]['D'],find_smae.iloc[0]['A1'])] =
cast_list[(find_smae.iloc[0]['D'],find_smae.iloc[0]['A1'])] + 1
        else:
            cast_list[(find_smae.iloc[0]['D'],find_smae.iloc[0]['A1'])] = 1

    colab_list = []

    for key,val in cast_list.items():
        a,b = key
        #print(key)
        #print(val)
        colab_list.append((a,b,val))
    colab_list = sorted(colab_list,key= lambda x: x[2],reverse = True)
    print("Top Collaborations:")
    #range of 5
    for x in range(5):
        print(colab_list[x])
    return colab_list

'''
b) Write a function called display_top_directors that displays the ranking of
movie
directors from the top grossing list ordered by the total box office money
they produced
```

```
'''
def display_top_directors():
    #read excel files
    top_casts = pd.read_csv('imdb-top-casts.csv',header=None)
    top_casts.columns = ['Title','Year','D','A1','A2','A3','A4','A5']
    top_grossing = pd.read_csv('imdb-top-grossing.csv')
    my_list = dict()

    for index, row in top_grossing.iterrows():
        title_of_movie = row['Title']
        money_generated = row['USA Box Office']
        find_director = top_casts.loc[top_casts['Title'] == title_of_movie]

        if (find_director.iloc[0]['D'] in my_list):
            my_list[find_director.iloc[0]['D']] =
my_list[find_director.iloc[0]['D']] + money_generated
        else:
            my_list[find_director.iloc[0]['D']] = money_generated

    top_directors_list  = []

    for key,val in my_list.items():
        top_directors_list.append((key, val))

    top_directors_list = sorted(top_directors_list,key =  lambda x: x[1],
reverse = True)

    for i in range(5):
        print(top_directors_list[i])

    return top_directors_list


    '''
    c) Write a main() function that tests the code from parts a) and b).

    '''
def main():
    display_top_collaborations()
    print('-' * 20)
    display_top_directors()

main()
```

screenshot:

```
In [103]: runfile('D:/COP4045 Python/Hw2/p4_Basantes_Charlie.py', wdir='D:/COP4045 Python/Hw2')
Top Collaborations:
('Christopher Nolan', 'Christian Bale', 3)
('Martin Scorsese', 'Robert De Niro', 3)
('Akira Kurosawa', 'Toshirô Mifune', 3)
('Charles Chaplin', 'Charles Chaplin', 3)
('Clint Eastwood', 'Clint Eastwood', 3)
--------------------
('Steven Spielberg', 3059836183)
('George Lucas', 1626418480)
('James Cameron', 1624021499)
('Peter Jackson', 1552388216)
('Christopher Nolan', 1479359328)

In [104]:
```